(4)

ISI Reprint Series
ISI/RS-88-204
March 1988

University of Southern California

Brian Harp
Robert Neches

NOTECARDS: An Everyday Tool for Aiding in Complex Tasks

Reprinted from *Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes,* held in Monterey, March 29, 1988.

INFORMATION SCIENCES INSTITUTE ISI
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT This document is approved for public release, distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RS-88-204	5. MONITORING ORGANIZATION REPORT NUMBER(S) ---------------
6a. NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute	6b OFFICE SYMBOL *(If applicable)*
7a. NAME OF MONITORING ORGANIZATION ---------------	
6c. ADDRESS *(City, State, and ZIP Code)* 4676 Admiralty Way Marina del Rey, CA 90292	7b. ADDRESS *(City, State, and ZIP Code)* ---------------
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Logistics Command DARPA	8b. OFFICE SYMBOL *(If applicable)*
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA903-86-C-0178 F33600-87-C-7047	
8c. ADDRESS *(City, State, and ZIP Code)* Air Force Logistics Command Wright Patterson Air Force Base Dayton, Ohio 45433-5001 DARPA 1400 Wilson Blvd. Arlington, VA 22209	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
---------------	---------------	---------------	---------------

11. TITLE *(Include Security Classification)*
NOTECARDS: An Everyday Tool for Aiding in Complex Tasks [Unclassified]

12. PERSONAL AUTHOR(S) Harp, Brian; Neches, Robert

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Research Report	FROM ______ TO ______	1988, March	28

16 SUPPLEMENTARY NOTATION
Reprinted from *Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes*, held in Monterey, March 29, 1988.

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
expert systems, hypertext, intelligent interfaces, notecards

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Most current notecard systems don't take advantage of the semistructured nature of notes, making retrieval and analysis difficult and requiring manual maintenance of the notes contained in the system. In addition, all current notecard systems are self-contained, forcing the user to maintain the relationships between notes and the domain knowledge bases. By exploiting the semistructured nature of notes, it is shown that notes can aid in several areas, including: capturing information not typically in intelligent systems, accessing that information, graceful degradation in intelligent systems, and reasoning using this additional knowledge. A system is proposed that contains structures for capturing semistructured information, a number of note types useful in reasoning in varied domains, and the control mechanisms necessary in a notecard environment. A prototype has been built and integrated with a knowledge base browsing tool, and is currently being expanded in capability as well as being integrated into a second domain.

[illegible]

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☑ UNCLASSIFIED/UNLIMITED ☑ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Sheila Coyazo Victor Brown	22b TELEPHONE *(Include Area Code)* 213-822-1511
	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ISI Reprint Series
ISI/RS-88-204
March 1988

University of Southern California

Brian Harp
Robert Neches

NOTECARDS: An Everyday Tool for Aiding in Complex Tasks

Reprinted from *Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes,* held in Monterey, March 29, 1988.

INFORMATION SCIENCES INSTITUTE
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported in part by the Defense Advanced Research Projects Agency under Contract No. MDA903-86-C-0178, and in part by the Air Force Logistics Command under Contract No. F33600-87-C-7047. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the AFLC, the U.S. Government, or any person or agency connected with them.

NOTECARDS: AN EVERYDAY TOOL FOR AIDING IN COMPLEX TASKS *

Brian Harp, Robert Neches

USC / Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
(213) 822-1511

1 Introduction

Paper notes are used for many tasks in our everyday life, whether it be a software professional using notes to remember bugs that need to be fixed or homemakers keeping track of chores to be performed. In intelligent computer systems, notes are potentially important because they can capture supplemental information about objects in a knowledge base (information not captured in traditional knowledge representation); for example, peripheral domain knowledge or interactions between a system and a user. This paper presents research in use of notes and the status of TINT (The Intelligent Note Taker), a medium for organizing and capturing such information in an electronic notes system.

Our research on the use of notes, and consequently TINT, is intended to address four key problems:

1. The lack of organization beyond ad hoc structure and references, imposed and maintained solely by the user's self-discipline, makes it difficult for users to keep track of their notes as the collection grows in size.

2. That same lack of organization means that large amounts of useful information contained in the notes is largely inaccessible to any system.

*The research described in this paper was supported by DARPA under contract #MDA-903-86-C-0178, and by the Air Force Logistics Command under contract #F33600-87-C-7047. The contents of this paper represent the views and conclusions of the authors, and do not reflect the policies or beliefs of the sponsoring agencies.

3. The lack of a mechanism foı recording semistructured knowledge, and communicating that knowledge to a system, limits users' abilities to guide intelligent systems that have gotten into boundary areas of their knowledge.
4. The lack of structure of notes limits the ability of an intelligent system to assist in sharing of notes between users.

The key feature that distinguishes our approach from other notecard systems is that it is based on a formal representation for capturing the content of notes and exploiting their semistructured nature. Also, because of the knowledge representation language being used, notes can be linked to other notes or knowledge base objects in an arbitrary fashion and the links are formally represented so notes can be easily found and reasoned about after their creation. Our goal is to provide a note environment that, when coupled with a domain specific knowledge base, will assume responsibility for the task of organizing, reasoning about and maintaining interrelationships between notes and note abstraction hierarchies.

This paper will first describe related research involving notes. Next, some of the supporting control mechanisms for an intelligent notes system will be discussed. Following that, note types will be presented and their use and functionality described, leading finally to a discussion of future work necessary to make notes a complete and domain independent computer tool.

Please note that the interface design issues discussed in this paper focus on conceptual interaction between the user and machine, as opposed to creation of interface objects such as menus and windows. The examples in this paper will be taken from two domains in which systems are being built at ISI. BEAMER (Bom[1] Expert Assistant for Managing, Evaluating and Reviewing) focuses on tools that aid Air Force material planners in stocking repair shops with the appropriate components in a timely manner. More generally, BEAMER will produce tools that aid users in the task of data review. The other domain is knowledge base and database browsing, maintenance and repair, where BACKBORD[Yen87] and TINT are integrated to help knowledge base developers in the task of building and maintaining knowledge bases/databases.

2 Recent Work

Note systems currently are both commercial products and objects of research. Xerox PARC has developed a commercial hypertext system [Halasz87] with a component called Notecards. This system allows a user to create and store notes. However, the representation of notes is very shallow, and it appears there is very little in this system that

[1]BOM stands for Bill Of Materials; BEAMER is being built in cooperation with Sacramento Air Logistics Center under funding from Air Force Logistics Command Headquarters.

actually allows reasoning about notes. Their system gives the user a template for a note and allows the user to fill in the information and store that note in a hierarchical file system. The user is expected to perform all actions on notes by hand (e.g., organizing and modifying). Because the notecard system is self-contained and not integrated with any knowledge representation system, the user must also manually maintain the relationships between notes in the notecard system and objects in the domain knowledge bases.

In most other hypertext systems, notes can be thought of as nodes. Nodes are pieces of text that are linked together by the user using linking facilities supplied by the system. It appears that the only organization capabilities (other than arbitrary linking of nodes) offered by these systems is that some allow the user to create node types. For example, the Design Journal developed in the MCC Software Technology Program, provides four node types for the designer to use - *notes, goals/constraints, artifacts and decisions*. These types provide no structural uniqueness to nodes, but help the user, by using a unique color or icon for each type, in differentiating the types of nodes they are working with. See [Conklin87] for a description of the Design Journal and other hypertext systems.

In contrast, our research has focused much more on representing notes in a way that allows reasoning about their semantics. Representation based on semantics allows a number of added capabilities:

- utilization of notes in applications in an integrated manner (see Section 3)
- efficient capture of knowledge (see Section 5)
- organization and reasoning about the notes (see Section 5 and Section 6).

Recent work done by Malone, et al.[Malone87] has given some insights into semistructured information that can be applied to a general notes environment. Their work has been focused on electronic mail messages, showing that the semistructured nature of mail is exploitable by using AI techniques for more explicit representation and reasoning. The research discussed in this paper is philosophically similar, but pushes Malone's ideas of semistructured information representation further. Our system is domain independent and suggests a generalization to a multipurpose knowledge-based approach for representing semistructured information.

3 The Impact of Notes in Intelligent Systems

This section will briefly discuss three implications of our approach on the design and use of intelligent systems. First, the problem of capturing outside knowledge will be considered - that is, information relevant to a system's reasoning that either was not anticipated in the system's design, or could not be formalized. Next, the problem of dealing with open worlds will be considered, in which systems need to take account of *outside knowledge to overcome the limitations of their own* knowledge. Third and finally, the issue will be raised of sharing knowledge, capabilities, and interface consistency across system components.

3.1 Representing Semistructured Knowledge

A note environment extends and aids traditional input of knowledge. Typically, the knowledge representation language used in a system limits the type of information that can be captured. For instance, in a frame-based system, a user can fill in attribute values of an object, add new objects and add new attributes. However, most frame based systems don't allow the user to input meta-information during execution of the system. This is because meta-information is assumed to be static and known during development of the system. Therefore, it is encoded into the knowledge base or the control mechanism and not directly accessible.

In the note environment, the user is free to attach information in the form of notes to any entity that the system formally represents, and the system can use that information in its reasoning in the future. For instance, if users acquire some relevant information about an attribute of an object, they can add a note that references that attribute. In the future, the user will have access to that information because it is linked in the knowledge base. (This is assuming that the knowledge representation language of the system captures these entities in a sufficiently rich way, as is the case with the NIKL[Kaczmarek86] and LOOM[McGregor87] knowledge representation languages used in TINT.) In LOOM, a concept roughly corresponds to a frame in a frame based system, and a role roughly corresponds to a slot in a frame based system. However, in LOOM, unlike most frame based systems, roles are concepts with a domain and a range role. This allows a system using LOOM to reason about roles just as it reasons about concepts. Thus, in the notes environment, notes can reference roles as well as concepts.

Notes allow the user to add knowledge that the system can reason about without having to express all of the information in as strict a formalism as is usually required in a traditional expert system representation. It is sometimes the case that the *existence* of certain types of information about objects in a knowledge base may be useful in reasoning, even though the information can be only partially represented in the system. Notes facilitate the capture of information that is not completely within the domain knowledge, yet has relevance to it in some way. They allow a system to utilize the fact that certain information exists to do further reasoning.

3.2 Graceful Degradation

Graceful degradation is a problem in most intelligent systems today. If peripheral knowledge can't be represented using the representation language of the expert system, or if some situation was not anticipated in the design of the system, it is simply not included in the system knowledge base. When the system reasons past the boundaries that this imposes, the system either does meaningless things or nothing at all. Notes will help this problem in that the peripheral information can now be captured and placed appropriately in the knowledge base, which means that a system can at least realize that it needs to collaborate with the user when it runs into these boundary situations.

An example from the BEAMER system should clarify this point. Consider the case where a material planner observes data indicating that a particular kind of circuit card has been used at a much higher rate than normal. The function of the BEAMER system is to help the planner[2] determine the causes of such conditions. Based on the findings, the planner can take action or at least know why the condition is taking place. Imagine that, in this case, the planner discovers that extra circuit cards are required from inventory because a high number of them are found to be faulty when they are installed.

BEAMER specializes in the data about components. This is necessary if BEAMER is to aid the user in the task of data review. However, it is not feasible for BEAMER to monitor the mechanics to see why the components are being used at such a high rate, nor is it feasible for the system design to anticipate all possible reasons or variants for this situation. This case is on the boundaries of the scope of the system; the system has a model of the problem, but not of all the causes of the problem. Therefore it must ask the user to handle the conditions involving causes that it doesn't have represented, as in this example. It should be noted that some domains can be modeled completely. However, even when this is feasible, it results in a very large representation, which may not be necessary.

Let us consider a related example, in which an excessive number of component **#11222**[3] is being required by the shop, and the system is trying to determine whether it should increase its predicted value for the **Future usage rate** of that component. There may be a number of reasons that component is being used excessively, all of which are outside the system's capability to infer. Assume that the predicted value should be increased if the component's mean time between failures has dropped, if the defect rate in replacement units has increased, or if past usage rates had been underreported. Any of these conditions will support a decision to increase the predicted rate. In this case, it is the *presence* of reasons, rather than the specific reasons themselves, that justifies

[2]The word "planner" refers to the *material planner* and should not be construed as a component of this knowledge base system.

[3]In this paper, knowledge base entities like objects and attributes will be in bold face type.

the increase.

Consider how this example would have to be handled in a conventional rule-based expert system approach. A conventional expert system would need to maintain a data structure representing a hypothesis, with various rules adding support or denial for that hypothesis. If the user had knowledge bearing on that hypothesis which was not captured by one of those rules, the only way to affect the behavior of the system is to manipulate the system's internal representation to modify the support for the hypothesis. In addition to the handicap of having to work in those terms, the user would have difficulty in finding a place to record the basis for the intervention, and so the system could not generate an explanation or justification of the outcome that would take account of the user's added knowledge.

On the other hand, since the information necessary to make this decision is that there is *justification* to increase the percentage, any information that justifies the increase is acceptable. If this system were reasoning with notes, any note that captured the justification for a increase is acceptable. Therefore, the system developer doesn't have to include in the reasoning all possible rules to support a hypothesis. In this example, attaching an **Increase usage rate note** to the appropriate field of the component could prompt the system to increase the percentage when it analyzed that component. In addition, the system can use that note as justification for the increase and the user, or any outside auditor, can see why the system took the action.

In the notes paradigm, if there is required information from outside the central domain, the system could provide the user with possible sources of information. The user would then have to retrieve the appropriate information (just as they have to do if a rule base is asking specific questions) and input it as notes linked to the appropriate objects. The system would use the note type and the structured content of the note to continue reasoning. In addition, after notes are in place and the expert system reasons down the same path in the future, it is in a position to recognize that its own knowledge was insufficient, and that an accurate conclusion will require collaboration between it and the user. Present-day systems are not capable of handling this, nor can they capture the connections between outside knowledge and their own domain knowledge[4].

The notes environment also enhances communication between a user and a system because it gives systems the ability to generate notes or partial notes. These notes will aid the user's understanding of the system and why the system is performing in a particular way. For instance, the BEAMER system may detect a discrepancy in data about a particular piece of equipment for which a planner is trying to ensure availability of parts. BEAMER will automatically create a note for that discrepancy and attach the note to the appropriate objects in the knowledge base. The note will contain some

[4]One of the rules for building successful expert systems is that the domain be bounded so that the knowledge of the domain can be "separated" from other knowledge. In practice this is usually difficult to do.

description of the discrepancy, probably a textual description of the formal constraint that was violated. However, the user can enhance this note by adding text or additional referents. In the future, the system may use the note by recognizing that it exists and, because of the note's type, refrain from further analysis until the note has been removed. In addition, the note is a reminder to the user to look into the problem and resolve it. Thus, notes provide a good intermediate representation that allows the system and the user the ability to work together, to keep track of their activities, and to create a more understandable model of the domain.

The system-generated notes described above are especially useful when there is the potential for a considerable number of messages being printed to the user at one time. For instance, in most systems today, when an illegal value is put into a slot in a frame, a message is printed to the user saying the value is illegal. In a system where there are a large number of constraints applied at once to data, there may be twenty or more messages at one time. The user cannot hope to resolve all of the problems that are generated by an action. Instead, one needs a system that marks the problems in a knowledge base and allows the user to address each problem at a time of their own choosing. System-generated notes are one answer to this problem (see Section 5.3 for a more detailed discussion).

3.3 Note Integration

As was mentioned in the introduction, a major benefit of the TINT notes system is that the notes are represented in the same knowledge base representation language as the domain knowledge bases. This allows users access to notes that have been created, as easily as they can access other knowledge base objects. Functionality defined for knowledge base objects will be as applicable to notes as it is to the other objects. In addition, notes and other knowledge base objects can reference each other in a completely integrated fashion.

Our note environment supports multiple-user integration, another useful aspect of integration. Notes capturing information specified by one user can easily be made accessible to other users if it is appropriate. For instance, in the example in Section 3.2 where excessive use of a circuit card was being investigated to decide whether to increase its predicted usage, the decision to increase was influenced by a note explaining that the defect rate had increased. It is quite common for items such as this to appear as components of several different kinds of equipment, including equipment handled by someone other than the material planner who may have recorded the original note. In the future, when knowledge bases are shared among a set of users, other materials planners will have access to that information and will be able to utilize it making decisions about other equipment items that have the same circuit card as a component.

4 TINT

TINT (The Intelligent Note Taker) is a prototype environment supporting the research in the use of notes. TINT currently doesn't recognize all of the different types of notes cited in the previous section, but has a representation of a generic note with the attributes specified previously and an interface that allows creation and manipulation of notes.

NIKL[Kaczmarek86,Robins86] [5] has been chosen as the knowledge representation and classification system for representation of notes. This choice was made for a number of reasons. The most important is that NIKL requires that objects are represented in formal and precise way. This allows NIKL to classify objects, organizing them on the basis of subsumption relationships. This is especially valuable when trying to recognize important features and group objects into classes and subclasses. Another advantage of using NIKL for representation of notes is that the note knowledge base will be completely compatible with other knowledge bases that have been developed for other domains. This integration is important because it means the notes are available at no cost to an applications developer; it comes with the development environment and is compatible with any other NIKL knowledge bases.

When notes are created, the NIKL concept representing the note is created and classified into the note network. The location of the concept in the network will depend on the referent or referents of the note, the creator of the note and the subsuming concepts specified for the note. The referents of a note also link the note directly to the appropriate concepts in the knowledge base.

Some key manipulation features of note systems have been identified and implemented. Figure 1 shows the TINT interface. The lower left form is a checklist for creating a note. It allows the user to fill in the attributes by typing in object names or choosing them from the knowledge base via assorted browsing mechanisms. The other forms surrounding the checklist in Figure 1 are being used to browse a knowledge base to find the referent for the note being created. This capability was integrated into the TINT system from BACKBORD ([Yen87], a knowledge base browsing and retrieval system developed at ISI). Here the user is selecting the *use-it* menu option on **Item.2840-00-037-9688TB**, which will make that object the **Referent** of the note being created. If the user had chosen to enter values from the keyboard, an editor would have been invoked in which to enter the information. When the user has completely filled out the note, they choose the *done* option at the bottom of the note-creation checklist. This executes a function that creates a concept representing the note, classifies it in the note hierarchy, and makes the appropriate links to the other knowledge base objects.

[5] LOOM[McGregor87], the successor of NIKL, will replace NIKL in TINT shortly.

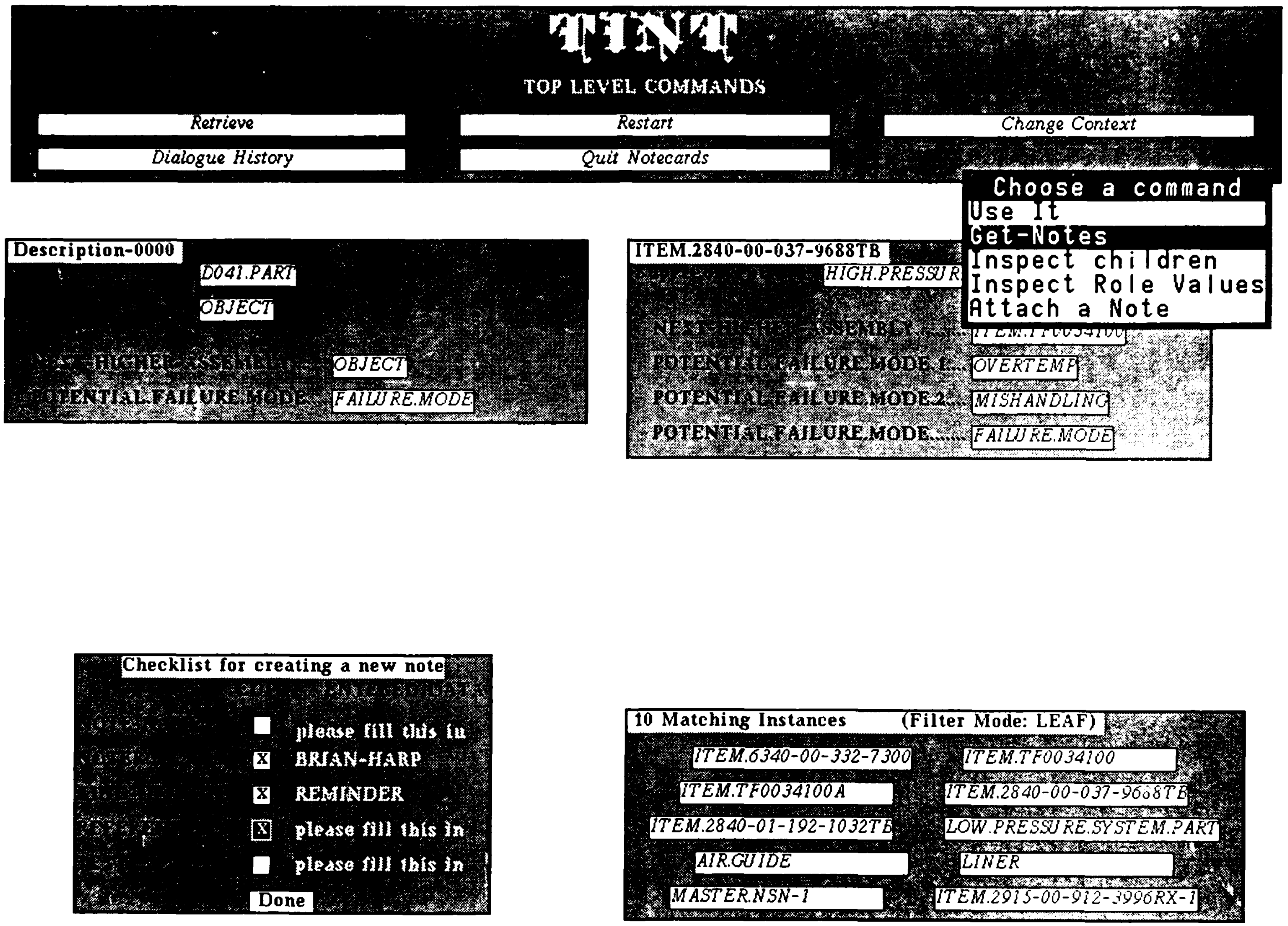

Figure 1: TINT Interface

There are also a number of ways to retrieve notes. A user can evaluate NIKL commands in a lisp environment to view concepts in the knowledge bases. Alternatively, the BACKBORD[Yen87] knowledge base retrieval system can be used to completely or partially specify descriptions of notes and then retrieve them from the knowledge base based on the attributes that the user cited. BACKBORD is based on the *retrieval by reformulation*[Tou82] paradigm and allows users to retrieve knowledge base objects by successively reformulating a description based on other objects.

Our experience with using BACKBORD has made us realize that being able to browse knowledge bases (as opposed to just retrieving information) for verification of construction, correctness of classification and consistency is very beneficial to knowledge base developers and knowledge engineers. Knowledge engineers often take notes about parts of the knowledge base that appear incorrect, listing errors that need to be corrected, etc. These notes are unorganized and usually sufficiently cryptic that only the person writing them knows what they mean. Based on this observation and the fact that the browsing capability was inherent in BACKBORD, we concluded that the notes facility would give organization and structure to the notes generated and that, when coupled with browsing capability, it would be a powerful tool for knowledge base development. This insight caused us to integrate TINT with BACKBORD so that the user could browse the knowledge base and add notes where necessary in the knowledge base.

5 Note Types

To understand the requirements for representing notes and reasoning about notes, a number of domains are being analyzed to determine common characteristics. It appears that notes, even though they are used for many different purposes, can be grouped by common attributes. These note types, defined by their particular attributes, can be used in a variety of domains. The idea of note types is not a new idea. As was mentioned in Section 2, some hypertext systems allow a user to define note types, primarily for ease of recognizing notes. However, notes types in these systems typically are not used in reasoning. The following sections will briefly discuss some of the note types and cite examples of how they are used in BEAMER and knowledge base browsing environments. Keep in mind that a note instance may have a number of these defined supertypes from which it inherits attributes and functionality.

There are a number of note attributes that will be referred to in this discussion. The attributes and their definitions are:

- **Notename** : the name of this note
- **Referent** : the knowledge base object(s) that the note references

- **Author** : the person or system that created the note
- **Content** : the body of the note, which may be either text or a structured data object.

These attributes define the root object of a note type-subtype hierarchy. This object will be referred to as a generic note in further discussion. Note that subtypes may differ from a generic note either by having more specialized values for one of these attributes or by having additional attributes.

5.1 Generic Notes

A **Generic note** is the most general note type, and all of the attributes of this note are inherited by its subtypes. This note can be used to capture any semistructured information that cannot be classified as a more specialized note type. The **Referent** of the note could be a class, an object or a list of classes or objects. Note also that, since notes are objects in a knowledge base, a note or class of notes could be a **Referent** value as well.

5.2 Demon Notes

A **Demon note** is a like a **Generic note** except that it has a **Condition** attribute. This attribute specifies the conditions for creation or linking of the note in a knowledge base. The conditions may either be defined for a type of note or for an instance of a note. If these conditions apply to a note type, then a new instance of that type is created each time the conditions are met. Of course, if the conditions are specified for an instance of a note, that instance already exists when the conditions are met.

Once the note instance exists, it is linked to the **Referents**, which are either specified by the creator or are derived from the conditions that made the note active. For example, when browsing knowledge bases, a developer may want a note to appear when there are more than five instances of a certain class of object. This note would serve as a reminder to inspect the instances and decide whether new classes are appropriate for those instances. Such a **Reminder note** would be defined as a subtype of **Generic notes** and of **Demon notes**.

Continuing with the example above, when five or more instances are classified under a class of objects, the **Reminder note** is created by the system and put in the knowledge base. The note will reference the class object and each of the instances that are of that class. In addition, the user will be shown, via the interface, that this note has been created. The user can then retrieve the **Referents** of the note and take appropriate action.

An example from BEAMER is as follows: A planner may want a note created to remind them to change the status of a component when it arrives in the material supply center. In this situation, if components for a piece of equipment cannot be obtained from a supplier, planners put that piece of equipment in a parts **Status** referred to as **Awaiting-parts**. This status is applicable until the parts arrive in the material supply center, at which time the status of the piece of equipment is set to **Active-repair**. There are a number of other actions that the planner must do when this condition is met, thus it is not appropriate for the system to take action without notifying the planner. If the planner has a **Demon note** defined to become active when a component's **Status** showed that it was no longer **Awaiting-parts**, the planner simply needs to input the **Status** of the parts and the system will remind the user of the consequences of the change of information. The planner may not be able to take immediate action on this situation, and the note will provide a visible message to the planner until they can perform the necessary actions.

5.2.1 Timed Note

A **Timed note** is a **Demon note** that has a specialized **Condition** attribute which refers to time only. It is similar to the *remind* facility in UNIX^{TM}. It will allow users to create notes that are to become active at some specified time in the future. One could imagine being able to specify notes that are displayed every year on January 1st or every month on the 15th.

5.3 Discrepancy Note

A discrepancy is defined as a constraint that has been violated. A **Discrepancy note** is a subtype of **Generic note** but its **Contents** attribute is restricted to be some form of description of the discrepancy that prompted creation of the note. In addition, a **Discrepancy note** is classified as a **Resolved Discrepancy Note** if it has a **Resolution** attribute that contains the response to the discrepancy, initiated by either the user or the system.

A **Discrepancy note** is created whenever a constraint has been violated. A number of constraint violation types exist. One type is a formal system constraint violation (e.g., the value restriction on an attribute is the constraint, and a violation of that constraint is that the wrong type of value is put in an attribute) which is generated by the system. Another type is a user-described constraint violation, where a user observes a problem in the system and creates a note to capture that problem. One can think of these problems as constraints that the user has on the system and that are violated by the current representation. A user-described violation is specified as uninterpretable text (uninterpretable by the system).

A **Discrepancy note** is defined as active until it is resolved, at which time it becomes inactive. The **Resolution** of a note can be specified in a number of different ways (see Section 5.4 on *resolution notes* below). It could be a note created by the user that describes the actions taken to resolve the discrepancy. Alternatively, the **Resolution** could be a system function or rule that resolved the discrepancy and prompted the creation of the **Resolution note**. In summary, formally representing **Discrepancy notes**, and their resolutions, allows the application system to manage discrepancies and problems, instead of making a user perform this task. Following is a discussion on both types of **Discrepancy notes**.

5.3.1 System Discrepancy Notes

Typically, in AI systems today, certain types of constraints can be defined and imposed on knowledge bases. However, if a constraint is violated, the message provided to the user is usually canned text. It is left up to the user to keep track of the violations and examine the knowledge base to understand what caused the violation to occur. This is done in a debugging style of interaction. The user must decide if the constraint is correct and there truly was an error in the data, or whether the constraint was incorrect and should be modified.

In systems where constraints are applied to large amounts of data, there may be a large number of discrepancies generated at one time and it is not feasible for a user to handle all of these potential violations immediately. For example, in the knowledge base browsing environment, when a new knowledge base is inspected by a developer and the knowledge base constraints are applied, there may be many problems in the knowledge base. The user would not find a list of error messages scrolling across the screen very helpful. It is much more useful if the system creates notes for each discrepancy and gives the user the capability to address the notes as they see fit. It may also be the case that a change made by the user could resolve a number of discrepancies. If the problems are just listed for the user and not related in any way to the knowledge base, there is no way to apply resolutions to multiple, related violations. Capturing these discrepancies in notes allows these relations to be exploited and explicitly represented.

This note type has a specialized **Content** attribute that requires the specification of the discrepancy to be system-interpretable. (More work has to be done to understand how tightly constraint violations and **Discrepancy note** can be integrated.) In addition, the **Author** attribute of the **System discrepancy note** is specialized to be the system.

5.3.2 User Discrepancy Note

This note type is the same as a **System discrepancy note** except that the contents of the **Discrepancy** attribute is textual (i.e., uninterpretable by the system). In addition, the **Author** attribute of the note is restricted to be a user and not the system. A situation in which a **User discrepancy note** would be used is shown in the following example: a developer may be browsing a knowledge base with *isa* hierarchies to verify consistency and debug existing problems. The developer may observe that an object is in the wrong part of the hierarchy and want to mark that object as needing action. This could be done using a **User discrepancy note**. The user can attach a **User discrepancy note** suggesting that the object be moved and is then free to continue browsing the knowledge base, knowing that he/she can come back to that note and move the object at a later date. In addition, this note will remain active until a **Resolution** has been specified for it. Accessing that note (and thus any of its **Referents**) can be accomplished by using the system-understandable attributes of notes (i.e., the note type, known classes of **Referent** value, etc.). The BACKBORD system is useful in locating objects in a knowledge base in this manner, and is described above in the discussion of TINT in Section 4.

User discrepancy notes can be resolved in the same way as **System discrepancy notes**. A user could create a note that specifies an action has been taken to resolve the discrepancy. Alternatively, the system may be able to automatically resolve a **User discrepancy note** if the system can reason about actions and their relation to the discrepancy. In the example above, the system may have a representation that allows it to detect when an object has been moved from the position at which the note was attached and conclude that that action resolved the "misplaced object" problem. It could then take action to remove the **Misplaced object note**. In this case, if the move occurs, the system will create a **Resolution note**, designate the move command (in its system interpretable form) as the **Resolution** value, and put the note in the **Resolution** attribute of the **Discrepancy note**. Of course, this type of note could be dangerous if the user was not notified before the system resolved the discrepancy, since inaccurate resolutions would be used to resolve problems that existed for other reasons.

The effect of putting the resolution in the **Discrepancy note** is that the **Discrepancy note** becomes inactive. One of the benefits of this approach is that the user can see the events that took place in creation and resolution of the discrepancy. In systems that just list the problems, no record of the events is captured for future use.

5.4 Resolution Note

A **Resolution Note** is used to resolve **Discrepancy Notes**. This type of note can be created either to capture the handling of a pre-existing discrepancy, or to anticipate discrepancies that may appear in the future. (In the latter case, the note type required is really a subtype of both **Resolution Note** and the **Demon Note**.) For instance, a user could create a resolution note that says, in effect: "if circuit card #112343 is found to be used excessively, the reason is because the cards are defective when brought into the inventory center. The data being collected on this part are correct". This note would become the **Resolution** of the **Discrepancy note** generated each time the data for that circuit card went out of bounds on a different piece of equipment. Again, this type of note can be dangerous if inaccurate or out-of-date resolutions are used.

A **Resolution note** has an added attribute which is called the **Discrepancy**. Its value is the discrepancy that it resolves; the purpose is to maintain a bi-directional link between the **Discrepancy note** and the **Resolution note**. Establishing the back-pointer from the discrepancy to the resolution also will cause the system to reclassify the **Discrepancy note**; it will then become a **Resolved Discrepancy Note**. In addition, the **Resolution note** should receive the same **Referents** as the **Discrepancy note** that it resolves.

There are a number of subtypes of **Resolution notes**, with the differences between them revolving around the kind of response and the **Author** of the response. For instance, for some kinds of discrepancies, the appropriate response is merely to generate an account that explains the cause of the discrepancy. In the data review task of the BEAMER system, as an example, material planners must investigate whenever it is determined that there is a discrepancy between the actual cost of materials used in overhauling an item and the cost estimate that was used to set the fixed price which customers are charged. It is often the case that the data on actual costs is accurate. Since Air Force practice is to fix the price of an overhaul job for two years at a time, and that price is then recomputed based on the assumption that the recorded data is accurate, all a materials planner can do in this case is to note that the discrepancy was investigated and record what factors had changed to invalidate the current cost estimate. Such a note would be an instance of a subtype of **Resolution Note** that we refer to as an **Explanation Note**. Such notes provide, as one of their side-benefits, an easy way for a system to filter out recurring discrepancies and refrain from re-presenting already-explained problems to the user.

Another subtype of note is an **Action Record Note**, which records an action taken in response to a **Discrepancy Note**. This in turn has several subtypes, such as **External Action Record Notes**, **User-initiated Action Record Notes**, and **System Action Record Notes**. The first of these three is used to record actions taken to resolve the problem that went outside the boundaries of the information system (e.g., fraud was suspected and the FBI was called); typically, these would be actions generated by the user, although the user might be following a system-generated recommendation in doing so. The latter two subtypes record actions taken within the scope of the information system, either by the user or by the system.

For an example of a **User-initiated Action Record Note**, consider the case discussed in Section 5.3.2, where a discrepancy was created during browsing because an object was found to be incorrectly placed in the knowledge base. In response to that discrepancy, the user might execute an interactive command to move that object to a more appropriate place. The record of that move having been accomplished constitutes the resolution of the original discrepancy.

System Action Record Notes serve to record actions taken automatically by a system. For example, in the current Air Force information system for material planners, called G005M, the user is given 90 days to make a revision when the actual usage rate for some component differs from its predicted usage. If the user takes no action by the end of that time, the G005m system automatically alters the predicted rate to correspond to the actual rate. No record is kept to indicate that the system has done so. This is unfortunate, because if the user wishes to review or question the system's actions at a later time, there is no convenient way of finding out which among thousands of database records the system has acted upon. In BEAMER, which is intended to provide a friendly interface between materials planners and systems like G005M, the system's action would be recorded in a note, and therefore would be retrievable later on.

The use of both user- and system- generated **Resolution notes** as a mechanism for recording history is potentially very important. There are a large number of situations in which it is crucial to have an "audit trail" that captures the relationship between concerns or observations and the conclusions or actions that resulted from them. This need is particularly strong in areas such as data review and analysis tasks which lead to decisions having significant financial costs or risks. We believe that in applications where the set of possible actions is reasonably bounded and well-understood, such as the BEAMER system, it should be possible for the application to maintain a model of the relationship between application-specific discrepancies and their possible resolution types. Such models can be used both to guide users in how to respond to discrepancies and to automatically detect when users have taken actions to resolve them, thus providing a higher quality of help and automatic maintenance of audit trails with no additional demands upon the user.

6 Note Manipulation Mechanisms

The notes system will have a number of support facilities as part of the note environment. One capability that is clearly needed is the ability to specify arbitrary criteria for retrieving and ordering notes. One use of that utility will be in creating lists and agendas of notes that the user can use in performing necessary tasks. For instance, a user may want to resolve problems in a knowledge base, fixing the oldest problems first. To perform this task, they may want a list of discrepancy notes ordered by date. This list could be generated by having the query utility first retrieve unresolved discrepancy notes and then order them according to date.

Computing the scope of a note is another mechanism that needs to be supported in a note environment. For example, one needs a way of specifying how a note is propagated when it is attached to a class of objects. In some cases, a note may be intended to reference a class object and not its children. In other cases, the note may be intended to be propagated to all children of that class.

A rich retrieval language is necessary in order to retrieve notes and other domain objects in an arbitrary way. Direct links are easy to support and they are handled by most frame or object systems. Other links are "indirect" (or inferred) references, and are more complicated because they relate objects in the hierarchy that do not explicitly reference each other, but that have some higher-level relationships. An indirect link is portrayed in the following example. Assume the user wants to retrieve from the knowledge base all notes that are relevant to John Doe, a mechanic with a particular area of expertise. To do so, the system must know how note objects relate to people objects. Assume also that the class of **People** is represented in the knowledge base and that **Mechanic** is a subtype of **People**. **John Doe** is an instance of **Mechanic**. In addition, **Mechanics** have an attribute that specifies their area of mechanical **Expertise** (e.g., F100 engines). The notes that are relevant to the **Expertise** of **John Doe** are the **Notes** that have the same **Referent** as the **Expertise** of **John Doe**. In other words, instances of **Notes** are retrieved if the **Referent** of the **Note** is the same as the **Expertise** of **John Doe**. This example describes an indirect relationship between **Notes** and **John Doe**. It is desirable for the retrieval system to be able to infer that notes relevant to **John Doe's Expertise** are relevant to **John Doe** himself. It seems unrealistic to believe that *all* indirect relationships can be *explicitly* stored in the knowledge base. For this reason the capability to compute relationships upon request is desirable.

7 FUTURE GOALS

The following are open areas of research that must be explored to complete a notes environment. The understanding of different types of notes and their characteristics is incomplete. A note epistemology is necessary to understand the role of notes in applications so that general note types for a note environment can be realized.

The user interface for notes is an area where empirical work must be done to develop an interface system that will allow the flexibility and generic note operations necessary to successfully move TINT to multiple domains. Because the notes represent knowledge that is very often at the perimeters of the domain knowledge, the notes interface needs to be flexible enough to relate notes to all aspects of a domain knowledge base, yet organization of the notes should be sufficient to allow a user to quickly gain access to notes from perspectives that are not fully realized in the domain.

Graceful degradation is a fourth area of research that will be aided by the development of notes. Paradigms will be investigated whereby notes can capture knowledge on the periphery of the domain of the expert system. Notes will be developed to capture the reasoning of the system and thus give the user a better understanding of what the system is reasoning about.

Finally, there is still much work to be done in characterizing the links between concepts, including both direct and indirect references. A language for specifying and reasoning about links between notes and application domains is an interesting area that could draw on related research in semantic nets.

8 Conclusion

This paper has argued for a paradigm in which notes are more than just personal reminders for an individual user. Over and above that use of notes, we view them as a medium for exchanging information between an individual user and other intelligent agents – including both humans and expert systems. In the framework described above, notes are represented as semistructured information objects. This means that they have principled formal semantics governing linkages between notes, and that individual notes have internal structure that is dependent on their classification in a type hierarchy. We refer to notes as *semistructured* because the specification method for their internal structure defines attributes (or, fields) for each note type, and may – or may not – require that all values of those attributes be represented in machine-interpretable form. Thus, systems can be built that are certain to understand some aspects of any note they encounter, without necessarily having to understand their entirety. We have identified four major deficiencies in present-day information systems which our approach is intended to address:

- Note systems today embody unprincipled approaches for organizing and interrelating notes, making it difficult for the user to manage their own note collections as they grow in size.
- Because of this lack of organization, there is no way for a system to analyze or interpret a user's notes, which means that they cannot help the user by organizing or augmenting notes, nor can they benefit from useful information locked within them.
- Intelligent systems today lack the ability to capture semistructured information and use it in reasoning about the domain.
- Because of the lack of structure in note systems today, it is difficult to allow sharing of notes between users.

The practical embodiment of our theoretical approach is TINT (The Intelligent Note Taker), a system under development which we are testing in several disparate applications. TINT will help address these problems by aiding users in a number of areas: explicit representation of semistructured information, graceful degradation of expert systems, and integration of semistructured information into expert systems. Compared to other note systems, our approach utilizes a much more formal knowledge representation language (NIKL/LOOM). This formal representation allows some unique control strategies that enable a system to use notes in reasoning past the boundary conditions of a system and allow a more coherent interaction between the user and a system.

Briefly, the specific features of TINT that address the four issues raised above are as follows. TINT represents specifications of note types and their attributes as concepts in the NIKL / LOOM knowledge representation languages. These languages have a formal semantics, classify new concepts according to those semantics, and manage inheritance accordingly. In TINT, notes are not directly linked to each other. Instead, they have a **Referent** attribute that links them to concepts found, or created, in an application-dependent knowledge base; connections between notes are determined from connections between concepts in the knowledge base. This enables the system to provide some very powerful tools for maintaining the organization of a system of notes: these range from classification-based reasoning to ensure consistency and completeness in establishing connections, to browsing and retrieval aids for finding and inspecting them.

The use of a formal representation language for notes also means that the internal structure of notes is specified in a manner which assures both that the meaning of attributes is represented and that the definitions of new note types are consistent with pre-existing types. This, in turn, means that structure has been provided to enable the system to analyze and interpret a user's notes; the system can therefore become involved in manipulating those notes, both by helping the user to organize them, and by understanding how to add notes of its own in a way that will fit into the organization and be meaningful to the user. Because the structure of notes are defined in a language with formal semantics in TINT, the system is in a position to reason about an individual note based on its place in a hierarchy of note types and the semantics of its attributes. This provides a basis for operating with partial understanding of notes, even in situations where the specific values of some of an individual note's attributes are not machine-interpretable, and thereby points to a resolution to the issue of enabling systems to deal with semistructured information.

We envision versions of TINT which access large knowledge bases shared by a number of users, although we have not yet implemented this aspect of our theory. When this comes to pass, notes associated with knowledge base concepts will become accessible to other users, as well as to the expert systems that might be performing tasks on behalf of those other users. TINT is indifferent to the source of notes in its mechanisms that ensure rigor, comprehensiveness, and comprehensibility in the placement and retrieval of notes. In a multi-user setting, we would expect it to continue to be effective in those respects. Thus, the approach provides a mechanism for ensuring that information entered by one user would be channeled to any other participating agents with need for it – without requiring that these needs be anticipated by a user or application developer.

Both TINT, and the model of semistructured information objects underlying it, are very much work-in-progress at this point in time. Both will be in a state of flux as to their specifics for some time to come. Nevertheless, progress to date encourages us to believe that this work represents a promising approach to providing a much more collaborative style of human/computer interaction than is possible today.

9 Acknowledgements

We wish to thank John Granacki for his positive and persistent critiquing of this paper.

References

[Conklin87] Conklin, J. 1987. Hypertext: An Introduction and Survey. In *Computer,* 20:17-41.

[Halasz87] Halasz, F., Moran, T., Trigg, R. 1987. Notecards in a Nutshell. In *Computer and Human Interaction and Graphics Interface Conference,* 45-52.

[Kaczmarek86] Kaczmarek, T., Bates, R., Robins G. 1986. Recent Developments in NIKL. In *Proceedings of the National Conference on Artificial Intelligence,* Philadelphia, PA, August.

[Malone87] Malone, T., et. al. 1987. Intelligent Information-sharing Systems. In *Communications of the ACM,* 30:390-402.

[McGregor87] McGregor, R., Bates, R. 1987. The Loom Knowledge Representation Language. In *Proceedings of Knowledge-Based Systems Workshop.*

[Robins86] Robins, G. 1986. The NIKL Manual.

[Tou82] Tou, F., Williams, M., Fikes, R., Henderson, D., Malone, T. 1982. RABBIT: an Intelligent Database Assistant. In *Proceedings of the National Conference on Artificial Intelligence,* 314-318.

[Williams84] Williams, M. 1984. What Makes Rabbit Run. In *International Journal of Man-Machine Studies,* 21, 333-352.

[Yen87] Yen, J., Neches, R., DeBellis, M. 1988. Backbord: Beyond Retrieval by Reformulation. In *Proceedings of Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes,* Asilomar, CA, March.